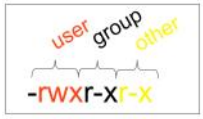
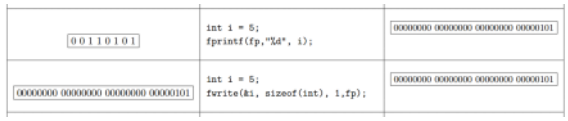
Review

March 27, 2016 11:43 PM

Week and Topic (Readings)	概括	难点，自己觉得不确定的，总结
Week 1: Jan 11-15 Admin and Shell	各种shell指令	琐碎的shell 指令需要记住： Sed, cat, grep(看是否match), cut(截取) File permission (e) Give the command to remove read permission for others (not in the group) from all the files without changing any other permissions. Remember that directories are files.  chmod • chmod 755 <filename> – 3 numbers between 0 and 7, the octal value for that category of user
Week 2: Jan 18-22 Introduction to C (King Chapters 1-7)	C 相关 重点： Arrays Scanf permission	Array: Char = 1 byte Int = 4 bytes 深刻理解C中的ARRAY： Must be in the same type 理解string literal 和 array的区别 Part (b) [3 MARKS] Show the output of each printf statement in the corresponding box. `char c[6] = "ABCDE";` `char *p = c;` `char *s = p + 2;` `printf("%c\n", p[0]);` A `printf("%s\n", p + 1);` BCDE `printf("%c\n", s[0]);` C `char *name = "John Tory";` `x = &name;` `y = *(name+3);` x: char** y: char Scanf: Permission: (worksheet) 1. Regular file and executable file? 2. Directories? 3. Remove read permission for others from all the files without changing any other permissions .
Week 3: Jan 25-29 Pointers, more arrays (King chapters 8,11, 12, 15)	**Pointers (难)** Types and Type Conversions	Pointers `int depth = 4; // Declares integer, sets its value to 4.` `int *p3 = &depth; // Declares pointer, points it to depth.` `int x; // Declares integer.` `x = *p3 + 5; // Sets integer to dereferenced p3, i.e. the value of depth, which is 4. Adds 5.` `depth = 1; // Sets depth to 1.` x= （not start with an address）… 意味着赋值，改变x的值，一次性行为 (*)x = & … 意味着points to，把x的memory address指向其他地方（一个地址），所以会一直跟着变 Get address => & Dereference => * a[i] and *(a + i) are equivalent in C 野指针: when an object that has an incoming reference is deleted or deallocated, without modifying the value of the pointer, so that the pointer still points to the memory location of the deallocated memory. Type conversion Casting long int strtol(const char *str, char **endptr, int base)
Week 4: Feb 1-5 Dynamically allocated memory, Strings (King chapters 13, 17)	Dynamic Memory Strings	Dynamic memory 存在heap里，而不是stack里，需要free Malloc时需要确定sizeof(type) Type *name = malloc(sizeof(type)*x); To allocate memory space for a string: malloc(strlen(str)+1). Null terminator Dangling pointers – Dereferencing a pointer after the memory it refers to has been freed. – Behaviour is undefined. Might: appear to work / bogus data / program crash Memory leak 比如malloc一个东西之后，忘了free就跳到别处去了，那个东西的memory永远无法得到，一直处于被占用状态 String: Remember that the storage needed for a string is one plus its length. // null terminator String 的 size 和 length 不相同。 length is the number of non-null characters currently present in the string. strlen gives the length of the string size if the amount of memory allocated for storing the string. malloc时要根据size String literal => 不能被修改，type为char* string系列function: strcpy The strcpy() function copies the string pointed to by src, including the terminating null byte ('\0'), to the buffer pointed to by dest. Strncpy The strncpy() function is similar, except that at most n bytes of src are copied. Warning: If there is no null byte among the first n bytes of src, the string

		placed in dest will not be null-terminated. If the length of src is less than n, strncpy() writes additional null bytes to dest to ensure that a total of n bytes are wri tten. Memory Model Code Fragment / Amount of memory / Where? / De-allocated when? int main() { char *name = "David"; — size of(char *) — stack - main — end of main/program 6 * size of(char) — read only memory — end of main/program char *c[5]; — 5 x sizeof(char *) — stack - main — end of main/program c[2] = name; c[3] = malloc(12); — 12 bytes — heap — end of main/program return 0; }
Week 5: Feb 8-12 Structs, Files, Streams (Kerrisk 4.1-4.6, 7.1.2, 7.1.3)	Structs and Union Streams Files Low-level I/O	Structs A collection of related data items a->b = (*a).b Struct is normally included in header file. typdef typedef struct MyStruct { int data1; char data2; } newtype; struct student good_student Streams stdin, stdout, and stderr are 0, 1, and 2 Stdin => function reads from (scanf), 通常可能是键盘，也可以通过redirection让A的stdin变成B的stdout。 Stderr => 另一个output渠道 Write a shell command that will run mktrans in the background, with no arguments, and will redirect standard output to a file called transactions and standard error to a file called err. SOLUTION: mktrans >transactions 2>err & Files Fread size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream) Ptr => fread会把东西读入ptr Size => 被读的每一个element的byte Nmemb => number of elements being read, each element has "size" Stream => pointer to a FILE object Return Value The total number of elements successfully read are returned as a size_t object. If this number differs from the nmemb parameter, then either an error had occurred or the End Of File was reached. Fwrite size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream) 和fread相反，fwrite把ptr里的内容，读入stream里 Fopen FILE *fopen(const char *filename, const char *mode) mode / Description "r" — Opens a file for reading. The file must exist. "w" — Creates an empty file for writing. If a file with the same name already exists, its content is erased and the file is considered as a new empty file. "a" — Appends to a file. Writing operations, append data at the end of the file. The file is created if it does not exist. "r+" — Opens a file to update both reading and writing. The file must exist. "w+" — Creates an empty file for both reading and writing. "a+" — Opens a file for reading and appending. wb, rb read/write binary Fclose int fclose(FILE *stream) fscanf int fscanf(FILE *stream, const char *format, ...) reads formatted input from a stream. This function returns the number of input items successfully matched and assigned fp = fopen ("file.txt", "w+"); fputs("We are in 2012", fp); fscanf(fp, "%s %s %s %d", str1, str2, str3, &year); fprintf sends formatted output to a stream. fp = fopen ("file.txt", "w+"); fprintf(fp, "%s %s %s %d", "We", "are", "in", 2012); read int read(int handle, void *buffer, int nbyte); The read() function attempts to read nbytes from the file associated with handle, and places the characters read into buffer. write int write(int handle, void *buffer, int nbyte); The write() function attempts to write nbytes from buffer to the file associated with handle. On text files, it expands each LF to a CR/LF. line feed => lf carriage return => cr fgets char *fgets(char *str, int n, FILE *stream) fgets reads a line from the specified stream and stores it into the string pointed to by str. It stops when either (n-1) characters are read, the newline character is read, or the end-of-file is reached, whichever comes first. Fseek Low level I/O binary files => fgets, fprintf, fscanf 都不能用，但是fread 和 fwrite还可以用

		char *fgets(char *str, int n, FILE *stream)  8bit = 1 byte 1 char => 1 byte 1 int => 4 bytes
Reading week		
Week 6: Feb 22-26 Binary Files, C odds and ends (Kerrisk 6.1-6.3, 6.6) (King Ch 14.1-14.4, 15)	**makefile** Useful C Features Function Pointers System Calls Errors and Errno	Makefile target: prerequisite1 prerequisite2 action1 action2 Make could have variable, and there are some special variable $@ the target $< First prerequiste $? All out of date prerequistes $^ All prerequistes CFLAGS = -Wall -g -std=c99 -Werror 在很多地方都需要的可以作为CFLAGS写在最前 gcc $(CFLAGS) <= 在action中以变量形式呈现 Pattern rule Most files compiled in the same way, write a pattern rule for the general case. %.o: %.c gcc ${CFLAGS} -c $< Clean 当所有都是最新时，就触发clean rm *.o <= 清除一切.o 文件 all: pre1 pre2 <= run make so that both programs are recompiled make xxxx looks for Makefile and looks for a rule with target xxxx and evaluates it Useful C features macro #define TAX_RATE 0.08 #define WITH_TAX(x) ((x) * 1.08) purchase = 9.99 WITH_TAX(purchase) typedef 参照struct Function pointers `double cm_to_inches(double cm) { return cm / 2.54; }` `int main(void) { double (*func1)(double) = cm_to_inches; char * (*func2)(const char *, int) = strchr; printf("%f %s", func1(15.0), func2("Wikipedia", 'i')); /* prints "5.905512 Wikipedia" */ return 0; }` 如果两个function有一样的signature (same return type, same number and type of arguments), 可以做一个function pointer `double time_sort(int size, void (*initializer)(int *, int)) { int arr[size]; initializer(arr, size); clock_t begin = clock(); insertion_sort(arr, size); clock_t end = clock(); check_sort(arr, size); return (double)(end - begin) / CLOCKS_PER_SEC; }` `int main() { srand(time(NULL)); for (int size = 1; size <= 4096; size *= 2) { double time_spent = time_sort(size, max_to_min_init); printf("%d: %f\n", size, time_spent); } return 0; }` System calls request service from the os I/O => read/ write <= scanf, printf use read/ write in the implementation `// File management system calls int open(const char *pathname, int flags, mode_t mode); ssize_t read(int fd, void *buf, size_t count); ssize_t write(int fd, const void *buf, size_t count); int close(int fd); // Process management system calls pid_t fork(void); pid_t getpid(void); pid_t wait(int *status); int kill(pid_t pid, int sig); int execv(const char *path, char *const argv[]);`

```
// File management system calls

    int open(const char *pathname, int flags, mode_t mode);
    ssize_t read(int fd, void *buf, size_t count);
    ssize_t write(int fd, const void *buf, size_t count);
    int close(int fd);

// Process management system calls
    pid_t fork(void);
    pid_t getpid(void);
    pid_t wait(int *status);
    int kill(pid_t pid, int sig);
    int execv(const char *path, char *const argv[]);

// Interprocess communication system calls
    int pipe(int pipefd[2]);
    int socket(int domain, int type, int protocol);
    int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
    int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
    int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
```

Errors and Errno

所有的system call 都用 perror查一遍，exit(1)
把system call的return value 和 0 或者 NULL 对比

如果不是system call
fprintf(stderr, "error msg: …")

Week 7: Feb 29-Mar 4
Midterm and Processes
(Kerrisk 24.1, 24.2, 24.4, 25.1, 25.2, 26
King 17.7)

Process Model

fork

exec

wait

fork

fork 时
only way to create new process (duplicate)

return value => if parent, return the actual pid; if child, return 0
getpid(); <= process's pid
getppid(); <= parent's pid

```
result = fork();

if (result > 0) {
    i = i + 2;       // parent process
} else if (result == 0) {
    i = i - 2;       // child process
} else {
    perror("fork");
}
```

wait

suspend the process itself to wait until its child terminates

pid_t wait(int *_wstatus_);

The call wait(&wstatus) is equivalent to: waitpid(-1, &wstatus, 0);
It would return a pid for the terminated children, -1 means still wait for child process

逻辑链

```
for (i = 0; i < 5; i++) {
        pid_t pid;
        int status;
        if( (pid = wait(&status)) == -1) {
            perror("wait");
        } else {
            if (WIFEXITED(status)) {// check if the process terminated
                printf("Child %d terminated with %d\n",
                    pid, WEXITSTATUS(status));// lower 8 bit exit signal
            } else if(WIFSIGNALED(status)){// check if temrinated by signal
                printf("Child %d terminated with signal %d\n",
                    pid, WTERMSIG(status));// return actual signal number
            } else {
                printf("Shouldn't get here\n");
            }
        }
    }
```

要理解这些macro的用法：

WIFEXITED(status) => This macro returns a nonzero value if the child process terminated normally with exit or _exit.

WEXITSTATUS(status) => If WIFEXITED is true of status, this macro returns the low-order 8 bits of the exit status value from the child process.

WIFSIGNALED(status) => This macro returns a nonzero value if the child process terminated because it received a signal that was not handled.

WTERMSIG(status) => If WIFSIGNALED is true of status, this macro returns the signal number of the signal that terminated the child process.

WIFSTOPPED(status) => This macro returns a nonzero value if the child process is stopped.

WSTOPSIG(status)

orphan and zombie

orphan process => child process 还没 terminates， parent process 已经 terminate 了
这种情况发生时，child process 的 parent 变为 init process with pid 1.

zombie process => child process 已经 terminates， 但parent process 还没有call wait 收集 child process的status
init process 会call wait 收集这些termination sstatus

execl

replace current process with a new executable, does not create a new process, just modify the process

```
int main() {
   printf("About to call execl. My PID is %d\n", getpid());
   execl("./hello", NULL);
   perror("exec"); // this line would be achieved if it weren;t able to load the program hello

   return 1;
}
```

execlp
int execlp(const char *file, const char *arg, ...);

execlp is a variable argument function. It takes 2 `const char *`. The rest of the arguments, if any, are the additional arguments to hand over to program we want to run - also `char *` - all these are C strings (and the last argument must be a NULL pointer)

So, the `file` argument is the path name of an executable file to be executed. `arg` is the string we want to appear as `argv[0]` in the executable. By

convention, `argv[0]` is just the file name of the executable, normally it's set to the same as `file`.
The `...` are now the additional arguments to give to the executable.

$ /bin/sh -c "ls -l /bin/??"

=> execlp("/bin/sh","/bin/sh", "-c", "ls -l /bin/??", (char *)NULL);

execvp

int execvp (const char *filename, char *const argv[])

The execv function executes the file named by filename as a new process image.

The argv argument is an array of null-terminated strings that is used to provide a value for the argv argument to the main function of the program to be executed. The last element of this array must be a null pointer. By convention, the first element of this array is the file name of the program sans directory names. See Program Arguments, for full details on how programs can access these arguments.

```
pid_t pid;
   char *const parmList[] = {"/bin/ls", "-l", "/u/userid/dirname", NULL};

   if ((pid = fork()) == -1)
     perror("fork error");
   else if (pid == 0) {
     execv("/bin/ls", parmList); // 第一个parameter是executable, in this case ls. 第二个parameter是传统意义上的argv, argv[0]等于executable
     printf("Return not expected. Must be an execv error.n");
```

Week 8: Mar 7-11
Processes (Fork, Wait, Exec)
(Kerrisk 2.11, 20, (21))

Pipes

Pipe

operate on file descriptors, communicate between process.
fd[0] => pipe for read
fd[1] => pipe for write

close the file descriptor you do not use at the beginning of process, close the file descriptor once finish use them

```
int main() {
    char line[MAXSIZE];
    int fd[2];
    // create a pipe
    if (pipe(fd) == -1) {
        perror("pipe");
    }

    int r = fork();
```

在程序最开始就建立pipe，在fork之前。

parent process

```
if (r > 0) {
    // Parent reads from stdin, and writes to child
    // close the read file descriptor since parent will write to pipe
    close(fd[0]); // 先把不要用的read pipe关闭

    printf("Enter a line > ");
    while (fgets(line, MAXSIZE, stdin) != NULL) { // 将内容从stdin写入line中

        printf("[%d] writing to pipe\n", getpid());
        if (write(fd[1], line, MAXSIZE) == -1) { // 将内容从line写入fd[1]中
            perror("write to pipe");
        }

        printf("[%d] finished writing\n", getpid());
        printf("Enter a line > ");
    }

    close(fd[1]); // 写完了，把write pipe也关了
    printf("[%d] stdin has been closed, waiting for child\n", getpid());

    int status;
    if (wait(&status) != -1)  { // wait系列
        if (WIFEXITED(status)) {
            printf("[%d] Child exited with %d\n", getpid(),
                    WEXITSTATUS(status));
        } else {
            printf("[%d] Child exited abnormally\n", getpid());
        }
    }
```

child process

```
 else if (r == 0) {
        close(fd[1]); //关了write pipe
        printf("[%d] child\n", getpid());
        // Child will read from parent
        char other[MAXSIZE];

        while (read(fd[0], other, MAXSIZE) > 0) { //用read把fd[0]中的内容读入other
            printf("[%d] child received %s", getpid(), other);
        }
        printf("[%d] child finished reading", getpid());
        close(fd[0]); //读完了，把read pipe关掉
        exit(0); //退出程序

    } else {
        perror("fork");
        exit(1); //时刻不忘查system call
    }
```

dup2

int fileno(FILE *stream);
The fileno() function shall return the integer file descriptor associated with the stream pointed to by stream.

change file descriptor means change stdin, stdout and stderr!

int dup2(int oldfd, int newfd);

The dup() system call creates a copy of the file descriptor oldfd, using the lowest-numbered unused file descriptor for the new descriptor.

The dup2() system call performs the same task as dup(), but instead of using the lowest-numbered unused file descriptor, it uses the file descriptor number specified in newfd. If the file descriptor newfd was previously open, it is silently closed before being reused.

This system call makes a copy of an open file descriptor. We will use it to reset the stdout file descriptor so that writes to stdout will go to our output file.

```
        if (dup2(filefd, fileno(stdout)) == -1) { // 得到filefd的file desciptor,并代替stdout的file desciptor，fileno专门输出file的descriptor
            perror("dup2");
            }
            close(filefd);
            execlp("grep", "grep", "L0101", "student_list.txt", NULL);
            perror("exec");
            exit(1);

            }
```

Week 9: Mar 14-18
Pipe
(Kerrisk 56.1 - 56.5, 59.2)

Signals

signal

kill
from shell => kill -SIGQUIT pid
from c => kill(pid, SIGKILL)

==========

struct sigaction (int signum, const struct sigaction *act, struct sigaction *oldact);

The sigaction() system call is used to change the action taken by a
process on receipt of a specific signal. acceptable values are defined in <signal.h>.

signum specifies the signal and can be any valid signal except
SIGKILL and SIGSTOP.

If act is non-NULL, the new action for signal signum is installed
from act. If oldact is non-NULL, the previous action is saved in
oldact.

```
struct sigaction {
        void    (*sa_handler)(int);
        void    (*sa_sigaction)(int, siginfo_t *, void *);
        sigset_t  sa_mask;
        int     sa_flags;
        void    (*sa_restorer)(void);
      };
```

sa_handler specifies the action to be associated with signum and may
be SIG_DFL for the default action, SIG_IGN to ignore this signal, or
a pointer to a signal handling function. This function receives the
signal number as its only argument.

Signal masks are used to store the set of signals that are currently blocked.
sigaddset(&sigset, SIGUSR2);

int sigprocmask(int how, const sigset_t *set, sigset_t *oldset);
how => how the signal would be modified
set => points to the set of signals to be used for modifying the mask
就是在随时随地把signal射程block 与 unblock

做题时：先写handler => 一个正常的function，该干嘛干嘛
Main 里面：install handler => struct sigaction => sa.handler => sa.flag => sigempty(&myName.sa_mask) => sigaddset? => sigaction(SIGINT, &myName, NULL)
先declare struct sigaction myNmae
Struct sigaction myName;
再把handler和之前的function连起来
myName.sa_handler = function;
设置flag
myName.sa_flag = 0;
设置一个empty set, make sure no sig is blocked:
sigempty(&myName.sa_mask);
改变sig 的action:
sigaction(SIGINT, &myName, NULL);
这样当function在运行时,收到之前设定的一些signal,就会根据handler走

```
void handler(int code) { // signal handling function, prints a msg to stderr
    fprintf(stderr, "Signal %d caught\n", code);
}
```

int sigemptyset(sigset_t *set);
The sigemptyset() function initializes the signal set pointed to by set

```
int main() {
    // Declare a struct to be used by the sigaction function:
    struct sigaction newact;
// Specify that we want the handler function to handle the
    // signal:
    newact.sa_handler = handler;
// Use default flags:
    newact.sa_flags = 0;
// Specify that we don't want any signals to be blocked during
    // execution of handler:
    sigemptyset(&newact.sa_mask); // create an empty set
// Modify the signal table so that handler is called when
    // signal SIGINT is received:
    sigaction(SIGINT, &newact, NULL); //当收到SIGINT时，按照newact的handler执行指令
// Keep the program executing long enough for users to send
    // a signal:
    int i = 0;

    for (;;) {
        if ((i++ % 50000000) == 0) {
            fprintf(stderr, ".");
        }
    }
return 0;
}
```

背

EXAMPLE 2

```
char name[20];
void sing(int code) {
  printf("Happy Birthday to you\n");
  printf("Happy Birthday to you\n");
  sleep(7);
  printf("Happy Birthday dear %s\n", name);
  printf("Happy Birthday to you\n");
}
```

```
void dance(int code) {
  printf("Let's dance!\n");
}
int main(int argc, char **argv) {
// since we don't know the length of argv[1], using strcpy is dangerous
    strncpy(name, argv[1], 19);
    name[19]= '\0';
// install handler for USR1 to call sing
    struct sigaction sa;
    sa.sa_handler = sing;
    sa.sa_flags = 0;
    sigemptyset(&sa.sa_mask);                                                    背
    sigaddset(&sa.sa_mask, SIGINT); // add signal SIGINT to the set &sa.sa_mask
    sigaction(SIGUSR1, &sa, NULL);  // 正常的流程，流程需要默出来
// install handler for USR2 to call dance
    struct sigaction sa2;
    sa2.sa_handler = dance;
    sa2.sa_flags = 0;
    sigemptyset(&sa2.sa_mask);
    sigaction(SIGUSR2, &sa2, NULL);
// make a set of signals that only contains USR2
    sigset_t sigset;
    sigemptyset(&sigset);  // create an empty set &sigset
    sigaddset(&sigset, SIGUSR2);   // add SIGUSR2 signal into the sigset
for (;;) {
      fprintf(stderr, "Who wants to dance?\n");
      sigprocmask(SIG_UNBLOCK, &sigset, NULL); // add signal in the set &sigset to the unblock
      // what happens if we get a SIGUSR2 now?
      sleep(10);
      sigprocmask(SIG_BLOCK, &sigset, NULL); // block the signal in the set sigset
      fprintf(stderr, "I'm too tired to dance\n");
      // what happens if we get a SIGUSR2 now?
      sleep(10);
   }
return 0;
}
```

Week 10: Mar 21-25
Signals
(Kerrisk 63.1 and 63.2.1)

Sockets

Socket

Client side
- socket -> connect

Server side
- socket -> bind -> listen -> accept

socket(AF_INET, SOCK_STREAM, 0)
- AF_INET => IPv4
- return a socket descriptor, <0 说明有问题

bind() assigns the address specified by addr to the socket referred to by the file descriptor sockfd.
int bind(int sockfd, const struct sockaddr *servaddr, socklen_t addrlen);
- sockfd => 刚刚socket 的return socket descriptor
- 建立一个server, server具有固定给的struct,需要设定几个参数（sin_family, sin_port, sin_addr.s_addr, sin_zero）

```
struct sockaddr_in server;
  server.sin_family = AF_INET;
  server.sin_port = htons(PORT_NUM);  // Note use of htons here
  server.sin_addr.s_addr = INADDR_ANY;
  memset(&server.sin_zero, 0, 8);  // Initialize sin_zero to 0
```

- convert a PORT number from host to network byte order
- The htonl() function converts the unsigned integer hostlong from host byte order to network byte order.
- The htons() function converts the unsigned short integer hostshort from host byte order to network byte order.
- The ntohl() function converts the unsigned integer netlong from network byte order to host byte order.
- The ntohs() function converts the unsigned short integer netshort from network byte order to host byte order.

int listen(int sockfd, int backlog)
- after calling listen, a socket is ready to accept connections
- prepares a queue in the kernel where partially completed connections wait to be accepted.
- backlog is the maximum number of partially completed connections that the kernel should queue.

int accept(int sockfd, struct sockaddr *cliaddr, socklen_t *addrlen);
- returns a new descriptor which refers to the TCP connection with the client
- reads and writes on the connection will use the socket returned by accept

server side

```
int main() {
// Create socket
    int sock_fd = socket(AF_INET, SOCK_STREAM, 0);// socket 完成
    if (sock_fd < 0) {
        perror("server: socket");
        exit(1);
    }

// Bind socket to an address
    struct sockaddr_in server;
    server.sin_family = AF_INET;
    server.sin_port = htons(PORT_NUM);  // Note use of htons here
    server.sin_addr.s_addr = INADDR_ANY;
    memset(&server.sin_zero, 0, 8);  // Initialize sin_zero to 0

    if (bind(sock_fd, (struct sockaddr *)&server, sizeof(struct sockaddr_in)) < 0) {
        perror("server: bind");
        close(sock_fd);
        exit(1);
    }

// Create queue in kernel for new connection requests
    if (listen(sock_fd, MAX_BACKLOG) < 0) {
        perror("server: listen");
        close(sock_fd);
        exit(1);
    }

 // Accept a new connection
    int client_fd = accept(sock_fd, NULL, NULL);
    if (client_fd < 0) {
        perror("server: accept");
        close(sock_fd);
        exit(1);
    }
```

Client side

socket -> inet_pton->connect

client只用设置family和port

```
int sock_fd = socket(AF_INET, SOCK_STREAM, 0);
struct sockaddr_in server;
    server.sin_family = AF_INET;
    server.sin_port = htons(PORT);

    if (inet_pton(AF_INET, "127.0.0.1", &server.sin_addr)< 0) {
        perror("client: inet_pton");
        close(sock_fd);
        exit(1);
    }
if (connect(sock_fd, (struct sockaddr *) &server, sizeof(server)) == -1) {
        perror("client: connect");
        close(sock_fd);
        exit(1);
    }
```

之后用read/write(sock_fd, "acutual content", size)就可以互相交流

The select() function indicates which of the specified file descriptors is ready for reading, ready for writing, or has an er ror condition pending. If the specified condition is false for all of the specified file descriptors, select() blocks, up to the specified timeout interval , until the specified condition is true for at least one of the specified file descriptors or until a signal arrives that needs to be delivered.

Week 11: Mar 28-April 1
Sockets
(King: 20.1, 20.2)

Bit Sets Shella

select

当一个parent process有多个child process时，可能会出现parent 等待一个child process 响应而不接受其他child read/ write的情况。使用select 避免这种情况发生？

int select(int nfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval *timeout);
nfds => largest file descriptors + 1
writefds => &fd_set
fd_set read_fds;
FD_ZERO(&read_fds);
FD_SET(pipe_child1[0], &read_fds);
FD_SET(pipe_child2[0], &read_fds);

int FD_ISSET(int fd, fd_set *set);
FD_ISSET() tests to see if a file descriptor is part of the set;

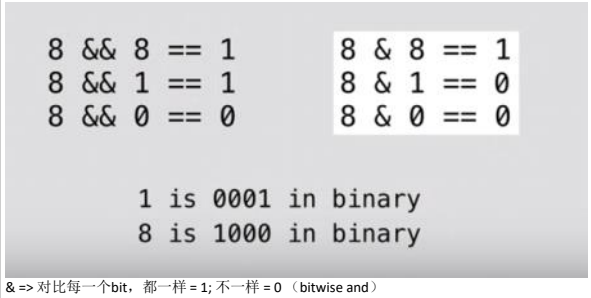
```
int main() {
    char line[MAXSIZE];
    int pipe_child1[2], pipe_child2[2];
// Before we fork, create a pipe for child 1
    if (pipe(pipe_child1) == -1) {
        perror("pipe");
    }
int r = fork();
    if (r < 0) {
        perror("fork");
        exit(1);
    } else if (r == 0) {
        handle_child1(pipe_child1);
        exit(0);
    } else {
        // This is the parent. Fork another child,
        // but first close the write file descriptor to child 1
        close(pipe_child1[1]);
        // and make a pipe for the second child
        if (pipe(pipe_child2) == -1) {
            perror("pipe");
        }
        // Now fork the second child
        r = fork();
        if (r < 0) {
            perror("fork");
            exit(1);
        } else if (r == 0) {
            close(pipe_child1[0]);  // still open in parent and inherited
            handle_child2(pipe_child2);
            exit(0);
        } else {
            close(pipe_child2[1]);
// This is now the parent with 2 children -- each with a pipe
            //  from which the parent can read.
fd_set read_fds;// prepare for the select
            FD_ZERO(&read_fds); // Initializes the file descriptor set fdset to have zero bits for all file descriptors.
            FD_SET(pipe_child1[0], &read_fds); //Sets the bit for the file descriptor fd
      in the file descriptor set fdset.
          FD_SET(pipe_child2[0], &read_fds);

int numfd; // prepare for the select, numfd = largest file descriptor + 1
            if (pipe_child1[0] > pipe_child2[0]) {
                numfd = pipe_child1[0] + 1;
            } else {
                numfd = pipe_child2[0] + 1;
            }
if (select(numfd, &read_fds, NULL, NULL, NULL) == -1) {
                perror("select");
                exit(1);
            }
// Read first from child 1
            if (FD_ISSET(pipe_child1[0], &read_fds)) { /we want to know if the pipe from child 1 is in read_fds after the select call. If it *is* in
the set, then we know that this pipe is ready, so we can read from it without fear of blocking. /
                if ((r = read(pipe_child1[0], line, MAXSIZE)) < 0) {
                    perror("read");
                } else if (r == 0) {
                    printf("pipe from child 1 is closed\n");
                } else {
                    printf("Read %s from child 1\n", line);
                }
            }
// Now read from child 2
            if (FD_ISSET(pipe_child2[0], &read_fds)) {
                if ((r = read(pipe_child2[0], line, MAXSIZE)) < 0) {
                    perror("read");
                } else if (r == 0) {
                    printf("pipe from child 2 is closed\n");
                } else {
```

```
                    printf("Read %s from child 2\n", line);
                }
            }
        }
        // could close all the pipes but since program is ending we will just let
        // them be closed automatically
    }
    return 0;
}
void handle_child1(int *fd) {
        close(fd[0]);  // we are only writing from child to parent
        printf("[%d] child\n", getpid());
        // Child will write to parent
        char message[10] = "HELLO DAD";
        write(fd[1], message, 10);
        close(fd[1]);
}
void handle_child2(int *fd) {
        close(fd[0]);  // we are only writing from child to parent
        printf("[%d] child\n", getpid());
        // Child will write to parent
        char message[10] = "Hi mom";
        write(fd[1], message, 10);
        close(fd[1]);
}
```

Week 12: April 4-8
Shell Programming, Review

Shell & bit manipulation

bitwise operator



& => 对比每一个bit，都一样 = 1; 不一样 = 0 （bitwise and）
&& => 只要数字不是0，就算作1
| <= bitwise or; ^ <= bitwise xor

shift operation
check a single bit


```
char var = 0xAE; // 1010 1110
a = var << 1;    // a becomes 0101 1100

char a = 0xAE; // 1010 1110
a = a << 2;    // a becomes 1011 1000

 a = 0x1;    // 0x01 (0000 0001 in binary)
 a = a << 1; // 0x02
 a = a << 1; // 0x04
 a = a << 1; // 0x08
 a = a << 1; // 0x10 (16 in decimal)


 a = 0x10;   // 0x10 (0001 0000 in binary)
 a = a >> 1; // 0x08
 a = a >> 1; // 0x04
 a = a >> 1; // 0x02
 a = a >> 1; // 0x01
```

How to check if a specific bit is 1 or 0

```
int main() {
    char b = 0xC1; // 1100 0001
         0x4       0000 0100

                   ---------
     b & 0x4       0000 0000
```

查看b&0x4的return结果，如果为1，就说明第三位（4的二进制为100）

Shell programming
不能乱加空格

variable
A=B <= 中间不能加空格
echo A <= 就是A

echo $A <= dollar 符号表示dereference, in this case B is the result.

```
bash-3.2$ CLASS=noon
bash-3.2$ echo $CLASS
noon
bash-3.2$ $CLASS
bash: noon: command not found
bash-3.2$ CLASS="noon"
bash-3.2$ $CLASS
bash: noon: command not found
bash-3.2$ COMMAND=date
bash-3.2$ date //特殊字符
Wed 30 Mar 2016 12:29:07 EDT
```

for loop

for
do
done

a="1 2 3 4"

for b in $a do echo $b done 1 2 3 4	for b in "$a" do echo $b done 1 2 3 4

if statement test can be replaced by [ ]

```
if test ! –d notes
then
echo not found
else
echo found
fi
```

BEWARE:
space matters
here

```
if [ ! –d notes ]
then
echo not found
else
echo found
fi
```

if then elif then else fi

test
ne => not equal; gt => greater than; lt => less than; le => less than or equal to

Test

The built-in command test is used to construct conditional statements in Bourne shell

-d filename	Exists as directory
-f filename	Exists as regular file
-r filename	Exists as readable
-w filename	Exists as writable
-x filename	Exists as executable
-z string	True if empty string
str1 = str2	True if str1 equals str2
int1 -eq int2	True if int1 equals int2
-ne -gt -lt -le	
-a -o	and or

quotes

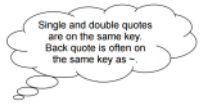
Quoting

Double quotes inhibit wildcard replacement only.

Single quotes inhibit wildcard replacement, variable substitution and command substitution.

Back quotes cause command substitution.

Practice and pay attention.



single

wildcard replacement
variable substitution
command substitution => 让具有command属性的字符恢复其command属性

" – double quotes
' – single quote
` - back quote

```
$ echo Today is date
Today is date
$ echo Today is `date`
Today is Thu Sep 19 12:28:55 EST 2002
$ echo "Today is `date`"
Today is Thu Sep 19 12:28:55 EST 2002
$ echo 'Today is `date`'
Today is `date`
```

Seq to loop
`seq 15`

#!/bin/sh

Positional parameters

set – assigns positional parameters to its arguments.

```
$ set `date`
$ echo "The date today is $2 $3, $6"
```

The date today is May 25, 2006

shift – change the meaning of the positional parameters

```
#!/bin/sh
while test "$1"
do
   echo $1
   shift
done
```

Notice the while loop

$0	Name of script
$#	Number of positional parameters
$*	Lists all positional parameters
$@	Same as $* except when in quotes
"$*"	Expands to single argument "$1 $2 $3"
"$@"	Expands to separate args "$1" "$2" "$3"
$1 .. $9	First 9 positional parameters
${10}	10th positional parameter

Shift => $1 变为 $2 变为 $3 …

Expr
用shell 的方式进行计算

```
x=`expr $x + 1`
y=`expr $x \* 5`  #need to escape *
```

Read

```
#!/bin/sh
echo "Enter your name:"
read fName lName
echo  "First: $fName"
echo  "Last: $lName"
```

		`` while read line do echo $line done < $file ``` Reads one line at a time from a file.